Programmable Data Planes meets In-Network Computing: State of the Art, Challenges, and Research Directions

Leonardo Gobatto, Pablo Rodrigues, Mateus Saquetti, Weverton Cordeiro, and Jose Rodrigo Azambuja

Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
email: leonardo.gobatto@inf.ufrgs.br

***Abstract*— Improving network traffic in networks is one of the concerns between networking researchers and network operators since the architecture of modern networks still faces challenges to process large data traffic without the cost of consuming a significant amount of resources not related to computing specifically. At the same time, network programmability has enabled the development of new applications and network services, from software-defined networking to domain-specific languages created to program network devices and specify their behavior. The development of programmable hardware and hardware accelerators like FPGAs, GPUs, and CPUs help this new paradigm go one step further. Using the artifact of programmability of these devices to solve problems, such as improve the processing of data traffic is the key of in-network computing. It offers the opportunity to execute programs typically running on end-hosts within programmable network devices already incorporated on the network, thus being capable of providing a reduction on the in-network processing load and requiring no extra cost, since operations can be concluded using a fewer amount of devices of the network and no extra device are needed. In this paper, we survey in-network computing, as well as we suggest classifying related works to in-network computing according to the hardware accelerator used. Also, we discuss challenges and research directions.**

***Index Terms*— In-Network Computing; Domain-Specific Languages; Hardware Accelerator.**

I. Introduction

Typically, network devices are implemented upon fixed-function chips, which means they have their set of functionalities defined by manufacturers and are built upon the premise that their functionalities are not changed or extended during the life cycle of the network device, offering limited flexibility to researchers and network operators to reconfigure its behavior. Furthermore, these devices generally are configured using a device-by-device approach, therefore, giving an idea of a decentralized control and making network management difficult.

The emergence of Software-Defined Networking (SDN) has allowed researchers and network operators to think about the idea of a network more programmable. With SDN, the network intelligence is decoupled in a control plane and a data plane to make the network more flexible, taking the control logic out of the network devices, and letting it in charge of the forwarding roles. Fig. 1 shows an overview of a SDN architecture. Applications are in a higher abstraction than the control plane and interact with it through northbound interfaces. In SDN, the control plane becomes responsible for specifying forwarding rules, such as specifies through the use of a SDN controller and an interface (southbound) for management the table entries of network devices operating on the data plane. This way, SDN becomes a vehicle that can make possible for network operators to manage network devices in a more centralized manner since the configuration of network devices are focused on a single entity.

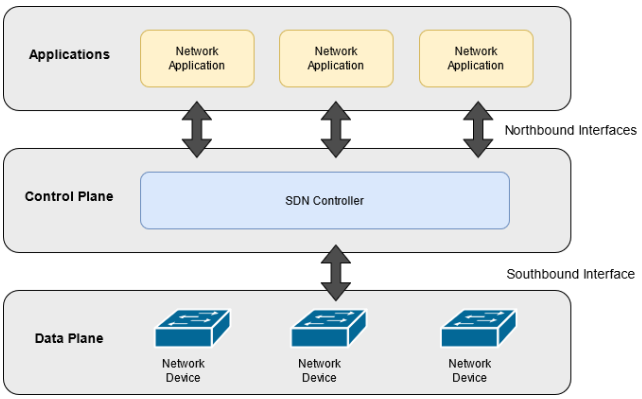


Fig. 1 A overview of the SDN architecture.

Despite SDN has improved some aspects to networks and network devices through advances in the control plane, concerns related to the data plane, such as how to extend the set of protocols which a network device operates on, as well as how to modify the forwarding algorithms of the device, or in other words, how a device must realize the data processing in hardware-level, is untapped by SDN. Therefore, ones are still faced with the limit of operating network devices based on the forwarding functionalities predefined by manufacturers, impeding the support to deploy and program a device with newer and novel protocols, rather than buy a new network device whenever new protocols initially not supported by a specific network device are enable to be used in networks. Advances in the control plane answer the question of "how to improve the network management?", but it lacks in answer the questions "how to extend the functionalities of a network device and how they must processing data?".

In the same way that advances have occurred in the control plane, advances in the data plane occurred as well. The advent of Reconfigurable Match-Action Tables (RMTs) [1], made feasible the ideia of programmability in the data plane. RMTs allow network operators and researchers to use a set of pipeline stages, each of them containing a match table of width and depth arbitrary, to match a certain field in an incoming packet on the pipeline. With this in mind, Programmable Data Planes (PDPs) emerged as new research area to be explored by the networking community. PDPs enable network devices to perform complex operations on

Digital Object Identifier 10.29292/jics.v16i2.497

packets, thus allowing the modification in how the data are processed on hardware-level, besides through Domain-Specific Languages (DSLs) created for PDP, such as POF [2] and P4 [3], allow programming the data plane behavior in a higher-level abstraction. DSLs provided novel approaches in PDP [4], helping to overcome even more the "ossification" of computer networking, and allowing the development of a new generation of network services.

In practice, network operators use programmable devices to modify the functionality of the data plane as needed while maintaining the ability to interpret and process data flows at link speed. The idea of employing the data plane for routing and processing flows is an old one, which goes back to the concept of Active Networks [5]. However, until more recently, the data processing capacity in the data plane was below the needs of the applications, which comes a bottleneck in the network.

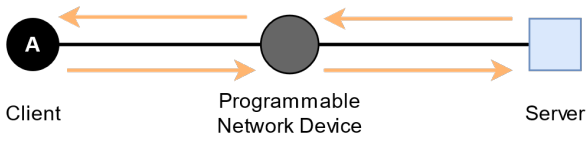
In this context, with advances occurred in the data plane, In-Network Computing arises as a new idea, which brings computing services to within the network. The release of a new generation of programmable network hardware and accelerators on the market [6, 7, 8, 9, 10, 11], and the exploring of the use of SmartNICs [12], Graphics Processing Units (GPUs) [13] and FPGAs [14], are making computing functions within network feasible, since they enable processing flows with high performance and low latency. Use cases like data caching, load balancing, DNS server, and data aggregation can be implemented within a network device, reducing the traffic in the network and consequently improving the overall performance.

In this paper, we present a survey on In-Network Computing. We aim for three objectives: (i) Give a background and present In-Network Computing to the reader; (ii) Offer a new taxonomy for the related works using hardware accelerators; and (iii), give to the community some insights about challenges and research directions in this area.

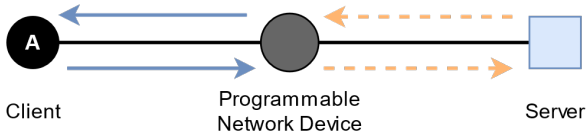
Below, we clarify as the paper is organized. The motivations for in-network computing functions using hardware accelerators are present in Section II. We present a background in Section III to familiarize the reader with common terms related to In-Network Computing. In Section IV, we discuss the state of the art, which includes the main existing researches related to the use of hardware acceleration for in-network computing functions. In Section V, we give insights into the challenges and research directions on In-Network Computing. Finally, we close the paper in Section VI.

II. Motivation

The literature has become vast in solutions that aim to decouple part of the intelligence of the application from a central point of a network and distribute it in devices that support the programmability of the data plane. Distributed systems rely more and more on memory, not hard drives. Therefore, the performance of system-based distributed applications is increasingly limited by network resources that do not compute [15]. In this process, routing devices are used to develop solutions that help to reduce network traffic and relieve congestion [16]. Network congestion is a major cause of performance degradation, impacting many applications that are sensitive to increased latency and packet loss [17]. When



(a) Traditional forwarding process.



(b) Forwarding process with computation within the network.

Fig. 2: Difference between traditional forwarding process and the forwarding process using in-network computing.

performing computing within a network, it is possible to reduce the delay and increase the throughput to improve the performance of the applications. As in-network computing applications become more complex, they require the use of hardware accelerators, since the computing time on routing devices cannot be delayed much longer than expected by the routing process. For instance, in Fig. 2(a), when a client *A* requests information contained in the server, the network device receives and forwards the requested packet to the server which will perform the required computation and replies with the result back to the network device, and finally, the network device returns the requested data to the client *A*. Through in-network computing, this computation can be coupled and performed in a programmable network device using acceleration, thus reducing the amount of data sent to the server and its workload. Fig. 2(b) shows an example of this approach. That is it, the capability of in-network computing enables the processing of network flows to be terminated as it travels across the network, rather than reach end-hosts, such as servers.

Although a set of In-Network Computing applications using hardware accelerators have been proposed (see Section IV), its full potential was not tapped yet. Concerns like programming abstraction, virtualization, and network architectures for support network hardware using acceleration to implement in-network computing applications are necessary (see Section V) in order to improve network performance and extract the maximum of the performance from resources available on each platform like FPGAs, GPUs and CPUs.

III. Background

In this section, we discuss the fundamental topics that underlie In-Network Computing, including DSLs for programming network devices, the Protocol-Independent Switch Architecture (PISA), which is an architecture based on the concept of programmable match-action pipelines, and FPGAs.

A. Domain-Specific Language (DSL)

A domain-specific language, is a computer programming language of limited expressiveness, focused on a specific domain. There are four key elements in this definition: (i) Computer programming language: A DSL is used by humans to instruct a computer to do something. As in any modern programming language, its structure is designed to facilitate understanding by human beings, but it must still be something executable by a computer; (ii) Natural language: a DSL is a programming language and, as such, must have a sense of fluency in which expressiveness comes not only from individual expressions but also from the way they can be composed; (iii) Limited expressiveness: a general-purpose programming language provides many features, such as support for varied data structures, control and abstraction. All of this is useful, but makes learning and use difficult. A DSL supports the minimum resources required to support your domain. One cannot build an entire software system on a DSL; Rather than, you use DSL for a specific aspect of a system; and (iv) domain focus: a limited language is only useful if it has a clear focus on a small domain. The focus of the domain is what makes a limited language worthwhile [18]. The adoption of a DSL involves pros and cons, and working with this approach means finding a balance between them. The benefits of DSLs include: (i) DSLs allow solutions to be expressed in the language and level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify and often even develop DSL programs [19]; (ii) The programs are concise, largely self-documenting and can be reused for different purposes [20]; (iii) DSLs increase productivity, reliability and maintainability [21] and allow validation and optimization at the domain level [22]; and (iv) DSLs improve testability and maintainability [23]. However, according to [19], the disadvantages of using a DSL are: (i) The costs of designing, implementing and maintaining a DSL; (ii) The costs of education for DSL users; (iii) The difficulty of finding the appropriate scope for a DSL; (iv) The difficulty of balancing between domain specificity and general-purpose programming language constructions; (v) The potential loss of efficiency when compared to manually coded software; and (vi) the limited availability of DSLs [24]. Operators can use domain-specific languages to effectively program network behavior, from control plane applications to routing devices.

Below, we describe two important DSLs that allow data plane programming: POF [25] and P4 [3]. POF was designed to interoperate with OpenFlow [26] and allow greater flexibility in defining packet processing, while P4 provides a high-level abstraction model for defining protocol headers, parser and deparser logic, and a packet processing pipeline, to achieve the same main objective as the POF. Although there are other DSLs that allow data plane programming, P4 and POF are the only ones recognized by the Open Networking Foundation (ONF), responsible for promoting open network standards.

B. Protocol-Oblivious Forwarding (POF)

As one of the initial implementations of SDN, OpenFlow provides a powerful set of tools for network operators to program and manage their networks in a flexible way [27]. However, its nature is protocol-dependent and, therefore, still limits the programming in the data plane. Specifically, OpenFlow defines the match fields in the network device tables according to the existing network protocols (for example, Ethernet and Internet Protocol). Therefore, OpenFlow switches need to understand the protocol headers to analyze packets and perform the match, which can results in serious compatibility problems when new protocols attempt to add or remove header fields. Thus, it is desirable that the programmability of the network can be further improved, in order to offer a forwarding plane that is protocol-independent and can be dynamically reprogrammed to support new protocol stacks. Following this idea, the Protocol-Oblivious Forwarding (POF) was proposed, a new SDN technology that tries to decouple network protocols from packet forwarding and make the data plan reconfigurable and programmable [28]. More specifically, POF is an extension of OpenFlow that introduces a protocol-independent instruction set and allows a network operator to define the protocol stack and packet processing procedure in a much more flexible way than in the current specifications of the OpenFlow [29]. This extension was designed to allow match-action rules to refer to arbitrary fields in packet headers and to control table matching behavior. Header fields are referenced in table entries using tuples $\{$type, offset, length$\}$, where type is set to 0 to indicate a header field or 1 to indicate a metadata field. Metadata fields can contain arbitrary data and serve the purpose of transporting package information through the processing pipeline. The offset is the initial position of the field in relation to the first bit of the header. Finally, the length of the field is specified in bits [25]. The packet parsing process is performed through table-based redirection and specially defined decision tables. These tables correspond to the header fields that indicate the next type of protocol (for example, EtherType), and their entries are populated to redirect the packet. As an example, suppose two decision tables dedicated to routing Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6), respectively. A third table can be defined to correspond to the EtherType field and filled with two entries. One of them would have a corresponding value of 0x0800 and redirect the packet analysis, via Goto-Table instruction, to the IPv4 table, while the other entry would have a corresponding value of 0x86DD and redirect the analysis to the IPv6 table. Packet analysis is therefore analogous to OpenFlow, but with the ability to combine and modify arbitrary header fields [4].

C. Programming Protocol-Independent Packet Processors (P4)

Programming Protocol-Independent Packet Processors (P4) is a high-level language that provides a suitable abstraction model for PDP with the capability of specifying and program the data plane behavior of a network device. [3]. Its main objectives are reconfigurability, protocol independence, and target independence. Reconfigurability proposes that the analysis and processing of packets can be redefined by the controller. Protocol independence suggests that the controller can extract header fields by specifying the packet analyzer and a collection of match-action tables for header

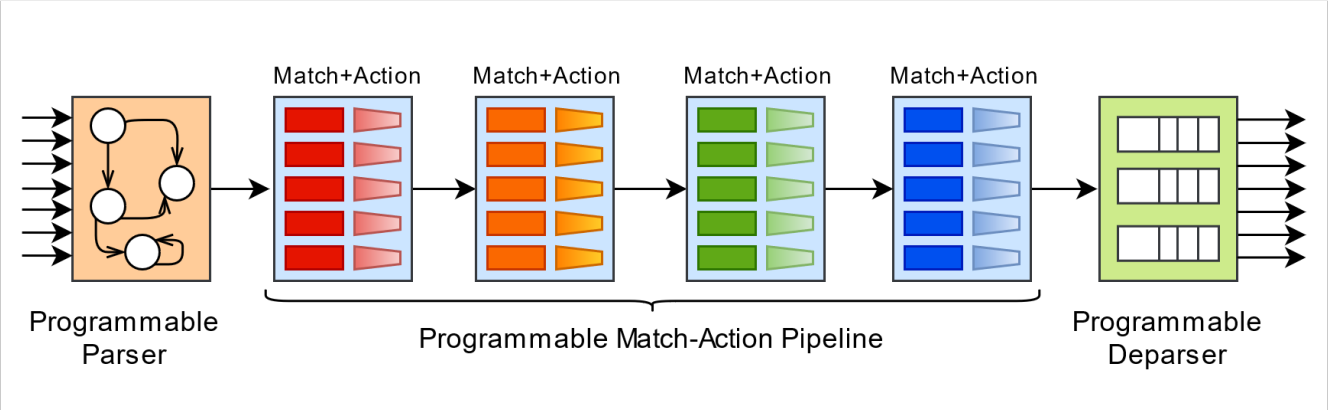


Fig. 3 The PISA basic pipeline.

processing. The target independence focuses on the possibility of the network operator specify the behavior of the switch while abstracts the non-relevant information about the target device. A P4 program mainly defines six items: (i) headers, which specify the names and widths of the protocol fields in which the program operates; (ii) metadata, structures that provide specific package information; (iii) parser, a group of state machines for analyzing headers and extracting data for metadata structures; (iv) match-action table, which identifies fields and metadata of packages to be compared and the possible actions being taken in response; (v) execution pipeline and control flow, to define how the packages are processed; and (vi) deparser, a process for rebuilding packages.

The task flow of the definition of a network device with P4 support begins with network operators writing and compiling a P4 program using a front-end compiler in a High-Level Intermediate Representation (HLIR). Then, a back-end compiler adapts the program to different targets, including FPGAs. Finally, through a runtime controller, operators can populate entries in the match-action tables in the data plane and allows that the destination device to process and forward packets. The main difference between these two DSLs is that POF is an extension of OpenFlow and P4 is a protocol-independent language, created to describe how packets are processed. Although POF is a direct evolution of OpenFlow, designed to interact with it and allow greater flexibility in defining packet processing, P4 has become popular in the industry and academia by providing high-level abstractions for defining protocol headers, header analysis and packet processing. However, both have the same main objective: to make the data plane programmable and reconfigurable.

D. Protocol Independent Switch Architecture (PISA)

Programmable switches allow implementing many complex network functions directly in the data plane. Protocol Independent Switch Architecture (PISA) uses the Reconfigurable Match Tables (RMTs) model, a pipeline architecture inspired by the Reduced Instruction Set Computer (RISC) for switch chips, by identifying the essential minimum set of action primitives to specify how headers are processed in hardware. RMTs allow data planes to be changed at runtime, without modifying the hardware. As in OpenFlow, the programmer can specify multiple match tables of arbitrary width and depth, subject only to a general resource limit, with each table configurable for matching in arbitrary fields [1]. The PISA architecture consists of a large number of physical pipeline stages to which a smaller number of RMT logical stages can be mapped, depending on the resource needs of each logical stage. Fig. 3 shows the stages of the processing pipeline in more detail. There are three fundamental elements, the Parser, the Match-Action Pipeline and the Deparser. The parser is implemented as a finite state machine that extracts and analyzes fields and headers of a packet. The match-action stages are composed of modules implemented in memory (for example, lookup tables (LUT), counters, and generic hash tables) and processing modules (for example, standard boolean and arithmetic operations, header modification operations, and hash operations), packages are recirculated through these stages until the end of processing. The deparser is a control block responsible for reassembling the headers of the packets on their way out.

The implementation of PISA is often prototyped in Application-Specific Integrated Circuits (ASICs) like in Barefoot Tofino ASIC [7]. It provides its users the ability to manufacture products with a proprietary design without having to initiate the design at the device level [30]. The language used to program these devices is P4, covering ASICs and processors that implement the PISA model [31]. However, the hardware implementation is not trivial and imposes restrictions on the number of bytes that can be examined in the packet header and the number of stages of match-action in a pipeline [32], since ASICs has immutable resources.

E. Field-Programmable Gate Arrays (FPGA)

The use of hardware-accelerated network devices in switches and routers is enabling rapid growth of the Internet. Currently, Ethernet switches are widely used to switch traffic on Local Area Networks (LANs) and to route protocol packets across Wide Area Networks (WANs). Commercial vendors use ASICs and/or FPGAs to speed up the switching, routing and processing of network data [33]. As previously explained, the new generation of SDN-related solutions introduced the notion of programmability of the data plane (for instance, P4 and POF). They allow faster development/provisioning of new protocols, as opposed to the long wait for the release of fixed-function ASIC switches that support standardized protocols [34]. The intrinsic characteristics of the FPGAs are aligned with the programming of the

data plane that seeks reconfigurability and programmability. More specifically, FPGAs are prefabricated silicon devices that can be electrically programmed to become virtually any type of digital circuit or system [35]. They offer several attractive advantages over fixed-function ASICs [36]. In addition, ASICs take months to manufacture and cost hundreds of thousands to millions of dollars to obtain the first device, while FPGAs are set up in minutes (and can usually be reconfigured if an error is made in generating a configuration file, commonly called bitstream) and cost from a few dollars to a few thousand dollars. The flexible nature of FPGAs, however, has a significant cost in area, delay and energy consumption: an FPGA requires approximately 20 to 35 times more area than an ASIC, it has a speed performance approximately 3 to 4 times slower than an ASIC and consumes approximately 10 times more energy [37]. These disadvantages arise in large part from the programmable routing mesh of an FPGA, which has a high cost in area, speed and power consumption to traffic data when compared to an ASIC.

An FPGA is composed of fundamental building blocks called Configurable Logic Blocks (CLBs), which provide the physical support for a design to be implemented and loaded into the FPGA, also, it contains Digital Signal Processing (DSP) slices and memory blocks. Fig. 4 illustrates the basic internal organization of an FPGA. The CLB is the main logical feature of the FPGA to implement sequential, combinatorial, and logical functions. Each CLB is connected to the routing loop through a routing matrix [38]. The DSP is designed to carry out digital signal processing functions, like multipliers, in a more efficient way than implementing those functionalities with CLBs. The referenced memory itself is a block RAM. The use of FPGAs allows fast prototyping but also takes advantage of flexibility, high performance, and reconfiguration. The main difference in the use of FPGAs, when compared to the use of microprocessors, is the ability to make substantial changes to the hardware, including changes in data and control flows [39], in addition to better efficiency in terms of performance and power consumption, since it implements a dedicated circuit. When compared to using a custom ASIC, the advantage of using FPGAs is the possibility to change the hardware at runtime, loading a different circuit in the reconfigurable matrix.

IV. State of the art

The emergence of programmable network devices has motivated the recent development of new research areas such as In-Network Computing. Placing small amounts of application computing on the network creates new challenges and opportunities to be explored. The state of the art of In-Network Computing classifies the applications implemented in programmable data planes based on their functionality [40] or behavior [41]. However, it does not have a clear taxonomy for the devices and hardware accelerators used. Therefore, in the following sections, we will briefly discuss the related works, classifying them according to the hardware accelerator used. Table I summarizes the related works using hardware accelerators on In-Network Computing.

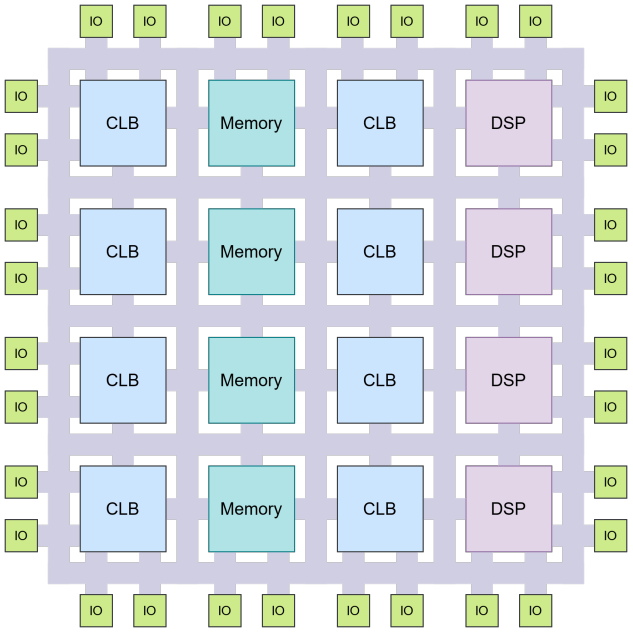


Fig. 4 Basic internal organization of FPGAs.

Table I.: Related works on In-Network Computing based on hardware accelerators.

Related Work	CPU-based	FPGA-based
IncBricks [42]	X	
Daiet [16]	X	
Cooke and Fahmy [43]		X
NetDebug [44]		X
Lake [45]		X
iSwitch [46]		X

A. *CPU-based Hardware Accelerators*

IncBricks [42] proposes a in-network cache “fabric”, with basic computation primitives, using programmable network middleboxes, named as IncBoxes, which are hardware units composed by a network accelerator and an Ethernet switch. The three essential functionalities of the middlebox design are: (i) Be able to parse in-transit network packets and extract the fields needed for the IncBricks logic; (ii) Be able to modify the header and payload and forward the packet given the hash of the key; and (iii), be able to caching the key/value data and perform some basic operations on the cached data. Its prototype reduced request latency by more than 30% and doubled throughput to 1024 bytes in a common cluster configuration. According to the authors, these results demonstrate the efficiency of network computing, and that efficient processing of requests in data center networks is possible if the computing is carefully divided between programmable switches, network accelerators, and end-users. However, these results depend on the capability of the programmer that develops the switch, since it can be difficult to utilize the full capacity without the hardware implementation details.

Daiet [16] is an open-source system based on CPU developed in P4 that performs in-network data aggregation. Its results show a wide rate of data reduction (86.9% - 89.3%) and a decrease in computing time. During the development of Daiet, was identified that P4 imposes implementation re-

strictions. When considering the frequency of the use of tables, since a table can be applied at most once per package, it is not possible to apply the same table to all headers in a stack of several headers of the same type. Additionally, deficiencies in data structures are founded, since the language does not provide variable-length data structures.

B. FPGA-based Hardware Accelerators

The work of Cooke and Fahmy [43] developed a model for evaluating the different approaches of in-network computing, that consider: the multiple levels of network structure; hardware differences and its changes on computing and networking, which includes accelerator platforms; realistic representation of metrics, such as performance, energy cost and financial cost. In order to test the proposed model they develop some Python scripts. This model is used to investigate a case-study scenario that demonstrates significant reductions in the latency of communication between devices, in addition to low computation latency when using acceleration in FPGA. An image classification application based on neural networks called SqueezeNet is used as a case study and quantifies the computation latency resulting from different platforms [43].

The NetDebug [44] proposes a fully programmable hardware-software framework for real-time validation and debugging of programmable data planes at full line rate. Its prototype was built on a NetFPGA-SUME using Xilinx SDNet, which translates P4 descriptions into hardware modules. Also, it is independent of the language of the determined application, validating data planes even in such a different workflow like high-level synthesis. The first assessment found that the state of the reject parser, an essential feature of the P4 language, is not implemented by SDNet.

LaKe [45] is a layered key-value storage design, executed as a network application. It works using multiple layers of cache, where each layer provides a trade-off between memory size and performance. This FPGA-based proposal achieves 17$\times$ better power efficiency than running on a host, with a transfer rate of 6.7$\times$ up to 13.6$\times$ higher, maintaining two orders of magnitude with better latency. Lake was implemented using Verilog, a hardware description language, on a NetFPGA-SUME board, but this language restriction can be impeditive for a programmer.

iSwitch [46] proposes a distributed solution using in-network computing to move gradient aggregation operations from network node servers to FPGA-based switches, reducing the number of network hops during gradient aggregation operations. Gradient aggregation is Reinforcement Learning (RL) operations used to train Artificial Intelligence (AI) applications. The results obtained by the study demonstrate that compared to the latest generation distributed training approaches, iSwitch offers an acceleration of up to 3.66 times for synchronous distributed training and 3.71 times for asynchronous distributed training while achieving better scalability. Also, the authors rethought the distributed RL training algorithms and propose a different hierarchical aggregation mechanism to increase parallelism and scalability.

C. GPU-based Hardware Accelerators

Graphics Processing Units (GPUs) are devices widely used on servers and very common in High-Performance Computing (HPC) applications. General-Purpose GPUs (GPGPUs), more specifically, transcend the domain of computer graphics and can be used to accelerate hardware in areas such as deep learning and cryptocurrency mining. Although GPUs are efficient for running heavy computing applications, they are usually not directly connected to the network, which makes them less suitable for network-intensive applications [47]. It is believed that this is one of the reasons that the in-network computing literature does not present many works that make use of these devices, however, it is difficult to assess how far network computing can go and what are the possibilities that the next generations of hardware for PDPs will make available.

V. Challenges and Research Directions

A concern observed in the literature is that when networking researchers and developers need to accelerate a certain computation in the network using programmable hardware, such as FPGAs, they need to master a set of secondary skills specific to the target, thus difficulting the development of the new solutions due to the extra knowledge needed. For instance, programming a switch using LaKe [45] requires extensive knowledge of the Verilog hardware description language. To address this gap, a solution that allows network developers and researchers to program network devices using popular DSLs, and to write down specific parts of the programs for acceleration (indicating, for instance, in the annotation, which architecture to consider for acceleration) is necessary.

The solution should abstract from the developers the specific details of the architectures to be used in the acceleration. A positive answer to this more fundamental concern will involve addressing some research and technological challenges, including (i) how to create abstractions of network function accelerators independent of the hardware architecture used in the acceleration, (ii) how to translate the codes written in a DSL and annotated for acceleration for the target hardware architecture, (iii) how to allow the dynamic connection and invocation of the software modules, running on the accelerators, with the programs running on the devices, (iv) how to allow data sharing between the program running on the network device and the software module running on the accelerator, (v) how to deal with aspects of synchronization and concurrent access to data shared between the accelerated module and the main program, and (vi) how to ensure performance similar to the scenarios that the developer writes his acceleration codes.

Virtualization and architectures for network hardware virtualization to maximize the use of in-network computing functions are needed as well, since virtualization has shown a powerhouse in computer networking, and specific architectures designed to use virtualization can potentialize its use. Sharing resources of hardware accelerators for computing in-network functions is necessary due to hardware accelerators must be used to implementing functions to reduce the congestion in the network with minimal use of resources as

possible, addressing how to share a set of resources between different modules running upon a hardware accelerator without a cost in performance. For instance, two virtual network devices running upon the same hardware substrate can share the use of the same computing function rather than allocate resources to implement the same computing function dedicated to each virtual network device. Also, researchers in respect to how the better way to share these resources is another research direction. Finally, we think that new in-network computing applications must be proposed, as well as researches related to what kind of applications are better improved when running as an in-network application. We expect that the community feels motivated to address these challenges and research directions.

VI. Conclusion

Handle with data traffic, link speed, and the use of resources is a key concern in a decade where people are more and more connected, thus requiring service suppliers to main a better performance to provide its services. Integration of devices like hardware accelerators with network devices to potentialize network performance is fundamental in this age. In this paper, we presented a brief survey on In-Network Computing focusing on the use of hardware accelerators. First, we gave motivations and present a background related to In-Network Computing. In a second moment, we classify the main related works in the area according to our new taxonomy, which classifies the works according to the hardware accelerator used in a more intuitive way. Finally, we present some challenges and research directions in this area. In any case, we expect that this survey motivates the community to research more about the usage of hardware accelerators in the context of In-Network computing.

Acknowledgements

The authors thank CNPq and FAPERGS for the financial support to this work. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This work also received funding from Grant #2020/05183-0, São Paulo Research Foundation (FAPESP). We also received support from Xilinx Inc.

References

[1] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 99–110.

[2] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 127–132.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[4] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 784–818, Oct 2017.

[5] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *Proceedings DARPA Active Networks Conference and Exposition*. IEEE, 2002, pp. 2–15.

[6] Marvell, *Prestera CX Data Center Switch*, 2019. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/switching/marvell-switching-prestera-98cx8500-product-brief-2019-03.pdf

[7] Barefoot Networks, *Tofino 2*, 2019. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino-2/

[8] Cavium, *LiquidIO II 10/25GbE Adapter family*, 2017. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/ethernet-adaptersandcontrollers/marvell-ethernet-liquidio-ii-cn23xx-10g-25g-product-brief-2017.pdf

[9] Marvell, *OCTEON III CN70XX and CN71XX Single to QuadCore Embedded Processors with Hardware Virtualization*, 2019. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-infrastructure-processors-octeon-iii-cn70xx-71xx-ap-produc/t-brief-2019.pdf

[10] Netronome Systems, *Netronome NFP-6000 Flow Processor*, 2018. [Online]. Available: https://www.netronome.com/m/documents/PB_NFP-6000_.pdf

[11] Digilent Inc., "Netfpga-sume reference manual," 2019. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/netfpga-sume/reference-manual

[12] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the ai accelerator?" in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, 2018, pp. 20–25.

[13] P. Sun, W. Feng, R. Han, S. Yan, and Y. Wen, "Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes," *arXiv preprint arXiv:1902.06855*, 2019.

[14] J. Woodruff, M. Ramanujam, and N. Zilberman, "P4dns: In-network dns," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–6.

[15] Huawei, "In-network computing for managed networks: Use cases and research challenges," *White Paper*, 2019. [Online]. Available: tools.ietf.org/id/draft-he-coin-managed-networks-00.html#JIN

[16] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 150–156.

[17] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 121–136.

[18] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[19] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," *ACM Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[20] D. A. Ladd and J. C. Ramming, "Two application languages in software production," in *USENIX Very High Level Languages Symposium Proceedings*, 1994, pp. 169–178.

[21] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton, "A software engineering experiment in software component generation," in *Proceedings of IEEE 18th International Conference on Software Engineering*. IEEE, 1996, pp. 542–552.

[22] D. Bruce, "What makes a good domain-specific language? apostle, and its approach to parallel discrete event simulation," *Kamin [43]*, pp. 17–35, 1997.

[23] E. G. Sirer and B. N. Bershad, "Using production grammars in software testing," *ACM SIGPLAN Notices*, vol. 35, no. 1, pp. 1–13, 1999.

[24] C. W. Krueger, "Software reuse," *ACM Comput. Surv.*, vol. 24, no. 2, p. 131–183, Jun. 1992.

[25] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 127–132.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, Mar. 2008.

[27] ——, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[28] S. Li, D. Hu, W. Fang, S. Ma, C. Chen, H. Huang, and Z. Zhu, "Protocol oblivious forwarding (pof): Software-defined networking with enhanced programmability," *IEEE Network*, vol. 31, no. 2, pp. 58–66, 2017.

[29] J. Yu, X. Wang, J. Song, Y. Zheng, and H. Song, "Forwarding programming in protocol-oblivious instruction set," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 577–582.

[30] A. K. Chan, J. M. Birkner, and H.-T. Chua, "Programmable application specific integrated circuit and logic cell therefor," Jun. 16 1992, uS Patent 5,122,685.

[31] T. Luinaud, T. Stimpfling, J. S. da Silva, Y. Savaria, and J. P. Langlois, "Bridging the gap: Fpgas as programmable switches," *GAs*, vol. 8, p. 10, 2020.

[32] T. Jepsen, D. Alvarez, N. Foster, C. Kim, J. Lee, M. Moshref, and R. Soulé, "Fast string searching on pisa," in *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, pp. 21–28.

[33] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "Netfpga–an open platform for gigabit-rate network switching and routing," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 160–161.

[34] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, "Dc. p4: Programming the forwarding plane of a data-center switch," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, pp. 1–8.

[35] I. Kuon, R. Tessier, J. Rose *et al.*, "Fpga architecture: Survey and challenges," *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[36] D. Chinnery and K. Keutzer, *Closing the gap between ASIC & custom: tools and techniques for high-performance ASIC design*. Springer Science & Business Media, 2002.

[37] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[38] S. Chandrakar, D. Gaitonde, and T. Bauer, "Enhancements in ultra-scale clb architecture," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 108–116.

[39] H. Yang, J. Zhang, J. Sun, and L. Yu, "Review of advanced fpga architectures and technologies," *Journal of Electronics (China)*, vol. 31, no. 5, pp. 371–393, 2014.

[40] T. A. Benson, "In-network compute: Considered armed and dangerous," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019, pp. 216–224.

[41] D. R. Ports and J. Nelson, "When should the network be the computer?" in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019, pp. 209–215.

[42] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "Incbricks: Toward in-network computation with an in-network cache," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 795–809.

[43] R. A. Cooke and S. A. Fahmy, "Quantifying the latency benefits of near-edge and in-network fpga acceleration," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 7–12.

[44] P. Bressana, N. Zilberman, and R. Soulé, "A programmable framework for validating data planes," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018, pp. 1–3.

[45] Y. Tokusashi, H. Matsutani, and N. Zilberman, "Lake: the power of in-network computing," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2018, pp. 1–8.

[46] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 279–291.

[47] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019.